# Learning graph rewiring

*Yahav Vanunu, Nitzan Leshem, Yaniv Hassidof*

November 2022

## Abstract

Oversquashing is a phenomenon in graph neural networks where information from too many nodes is condensed to a single vector, harming the network's predictions. Oversquashing is widespread in large networks where over the multiple layers of the GNN information from an exponential amount of nodes is condensed to the constant sized node embedding. It is shown that even simple solutions such as making the last layer of the GNN fully adjacent (adding all possible edges in the graph) improves task accuracy by up to 40%.

The goal of this project is to find and compare methods of graph rewiring to improve learning. Our algorithms will learn how to modify edges in such a way that will maximize performance across several tasks.

Our code is available here: Github.com/vYahav/GraphRewiring

# Introduction

Most conventional deep learning models demonstrate improved performance when additional layers are added. In contrast to these models, however, adding more layers in GNNs results in a decrease in performance. Each node in the GNN gains access to data from other nodes that are farther away when more layers are added. In theory, this should work great, but the results aren't what you'd anticipate. Two prominent effects that are unique to GNN are over-squashing and over-smoothing. Over-squashing is the distortion of information coming from distant nodes as a factor affecting the effectiveness of message transfer for tasks that are reliant on long-distance interactions. Over-smoothing is the inability to distinguish the embeddings of nodes of different classes.

In this project we will try to learn how to rewire the graph by adding and removing edges to assist information transfer using 3 different methods- Link prediction through Meta learning, Reinforcement learning and Minimum Spanning Tree.

**Reinforcement learning(RL)** is a machine learning method dealing with planning and dynamics. In the RL paradigm an agent interacts with an environment while receiving rewards. The goal is for the agent to learn the actions that will conclude in the highest rewards. The problem is usually formulated as a Markov Decision Process(MDP) where the current state of the environment is a node in a Markov graph and the actions along with the dynamics of the process dictate the transition probabilities towards the next states.

**Attention** in the context of deep learning is a mechanism for learning relations within the data to determine for each part of the data what information is more important and relevant to it given the context.
More specifically we use attention to learn embeddings for nodes/tokens that relate to one another such that nodes that are relevant to each other will receive higher similarity score, which we can later use for smart data aggregation.
In the context of Graph Learning the most common attention mechanism is used in the GAT architecture, where we learn a single embedding for each node and a fully connected layer that receives as an input the concatenated embedding of two nodes and its output is the attention score.

**Minimum Spanning Trees(MSTs)** are a subset of weighted undirected graph edges that constitute a spanning tree with minimal weight. For unconnected graphs we usually wish to find a minimum spanning forest, that is a collection of MSTs for each graph section. MSTs are widespread in classical graph problems that are related to efficient network connectivity. Examples include planning a rail line, a telecommunication network, etc. Since graph rewiring is about creating good network connectivity with minimal squashing, we assessed that MSTs are good candidates for edge subsets to wire.
A popular algorithm for finding MSTs is Kruskal's algorithm, that finds a tree in O(logV). The advantage of using a classical algorithm for such problems is providing a quick solution with no training required, even if the end results would not be tailored for each problem.

# Method

**Learn graph rewiring via Reinforcement learning:**

In our RL approach we formulate graph rewiring as a game where an agent has to decide which edges to prune\wire in order to achieve maximal performance on a given task. We chose the proximal policy optimization(PPO) algorithm to solve the task since it is a good general purpose algorithm. It requires little hyperparameter tuning and as such we preferred using it over other algorithms that might fit the task better but would require a large investment in tuning.

The PPO algorithm includes an actor network trained to choose the best action, and a critic network trained to help the actor with better predictions of the value of each state. Both of these networks are trained using gradient descent methods to find both the optimal policy and the approximate value function.

Since the possible state space is enormous for large graphs, we use a trained Graph Attention network to create a graph embedding. For the Cora dataset our node representation is a vector of length 8, such that the graph is represented by a 2078x8 matrix. Our actor network was configured to output 2 real continuous values (x,y), such that the edge between nodes (x,y) would be wired or trimmed if it already exists. We attempted using discrete actions as output, but the resulting actor network was too large to fit in our memory during training.

In our current RL formulation we modify the graph once and use it for all GNN layers. While we could modify the graph differently for each layer such an approach was not feasible with our current compute resources.

**Learn graph rewiring via attention weights:**

***Method I - One-shot Rewiring***

In this method we try to mitigate over-squashing by pruning edges with low attention scores as well as allow message passing between distant nodes by adding edges with high attention scores.

We trained a multi-headed two-layer graph neural network using its original edges on a specific task and then used its attention scores to prune and add edges (we averaged the scores over the attention heads to get a single score for each edge) and using the new edges we retrained the model from scratch.

Our pruning is done by defining the percentage of the edges we wish to remove and then removing the edges with the lowest scores until we reach our goal.

For edge addition we predefine an amount of edges to add and then add the edges with the highest scores until we reach our goal, we've also tried adding the single highest scored edge for each node instead to increase connectivity but this showed poor results.

We did the same rewiring experiment to both the first attention layer and the second layer together using each layer's attention scores, as well as each of the layers separately.

***Method II - Iterative Rewiring***

In this method we implemented iterative edge pruning and addition.

In each epoch we prune a small portion of the edges from the previous epoch and add a small portion of new edges according to the edges attention scores.

We do this for each layer separately (different graph connectivity for each layer).

We enforce that the number of edges we remove at each epoch is larger than the number of edges added, therefore the more epochs the algorithm runs the less total edges remain in the graphs.

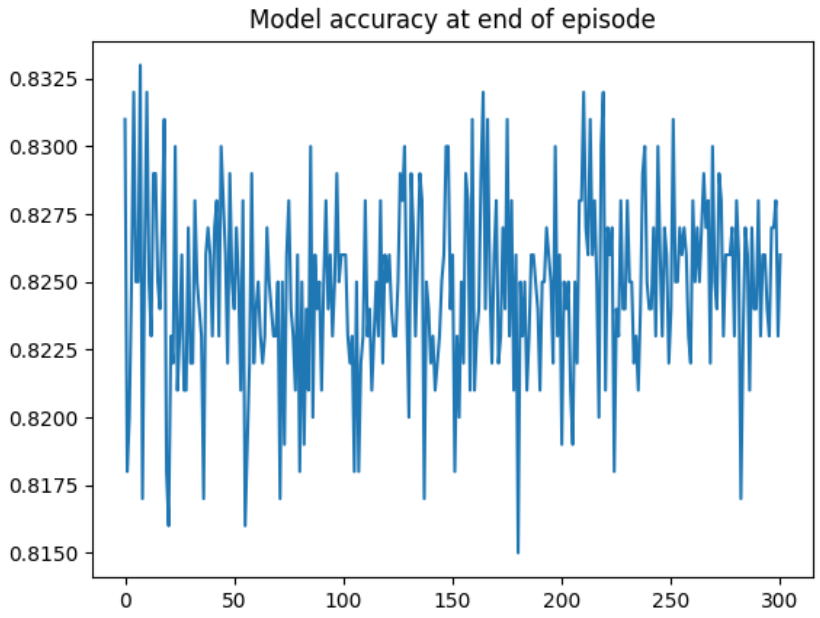
We begin with defining the start percentage of edges to prune or add and in each iteration we decrease both numbers.

**Learn graph rewiring via MST (Minimum Spanning Tree):**
Here, we tried rewiring the graph using MST. We first trained GAT on the Cora dataset to get baseline results to compare with later. Then, we ran Kruskal's algorithm on the Cora dataset and received the minimum spanning tree (or minimum spanning forest, in case the graph is not connected) of the graph. With this information, we tried 2 things- the first is to include only the edges of the MST which heavily diluted the graph (by more than 50%) and the second is to add the MST edges to the graph (which increased the amount of edges by ~50%). Then we trained the GAT again on the new graphs and examined the results.

# Results

When running the GAT model on the Cora dataset with the vanilla GAT network we got a baseline result of 82% accuracy.

**Graph rewiring by Reinforcement learning:**
On the Cora dataset our RL agent achieved 83.3% accuracy, showing it was able to find a rewiring with reduced information squashing. The training lasted 300 episodes, while in each episode the agent took 1000 actions. Training took 1 hour on our RTX 2070 card. The graph does not show a convergence, indicating that perhaps longer training will yield better final results.



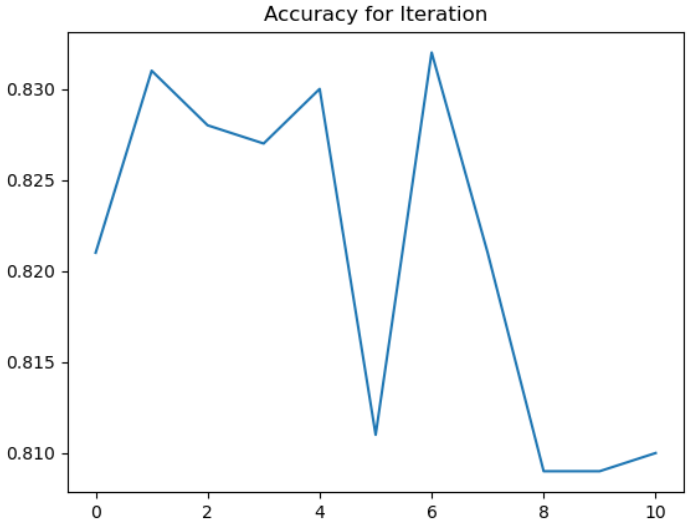
**Graph rewiring via attention weights:**

We ran the following three tests for each layer separately and both layers together. We ran all of the tests 10 times and averaged over the results.

One-Shot Rewiring-

| Model | Average Accuracy | Max Accuracy | STD |
|---|---|---|---|
| Base | 0.825 | 0.832 | 0.004 |
| First_Layer_Prune | 0.816 | 0.837 | 0.01 |
| First_Layer_Add | 0.817 | 0.827 | 0.007 |
| First_Layer_Both | 0.825 | 0.832 | 0.005 |
| Second_Layer_Prune | 0.822 | 0.832 | 0.006 |
| Second_Layer_add | 0.816 | 0.826 | 0.007 |
| Second_Layer_Both | 0.824 | 0.833 | 0.005 |
| Both_Layers_Prune | 0.821 | 0.832 | 0.007 |
| Both_Layers_Add | 0.812 | 0.827 | 0.008 |
| Both_Layers_Both | 0.818 | 0.831 | 0.01 |

Iterative Rewiring-



Our results indicate that pruning edges using attention scores could be effective in reducing the number of edges while keeping the same accuracy and sometimes better, while adding edges using weights is less effective, most likely due to the GAT architecture which in contrast to the transformer architecture does not use similarity to measure the score between embeddings, instead it learns a FC layer to calculate a score from the concatenated embedding, this can result in noisy and unpredictably results when measuring the score between nodes that did not originally had an edge, since the FC layer did not receive them as an input in training time

**Graph rewiring by MST (Minimum Spanning Tree):**
After finding the MST, and updating the graph edges to include only the MST's edges, we received 77% accuracy.
The result clearly indicates a reduction in accuracy (-5% from baseline) with this method. But on the positive side, with the reduction of edges to train on comes an improvement in training time- we noticed a 20-25% improvement in training time.

Also, when adding the MST edges to the existing graph edges we received accuracy of 75% (-7% from baseline), which is much worse accuracy and has no other benefits. We would not recommend doing this.

## Conclusion

We have seen that there were improvements in some of the methods (improved accuracy), and deterioration in others (decreased accuracy, sometimes with other benefits such as improved training times). There is still much research needed to be done in order to achieve major improvements in accuracy results with graph rewiring.

## Future work

For the RL approach the AlphaZero algorithm by deepmind shows great promise, with its variant AlphaTensor being used to discover the fastest ever matrix multiplication algorithm. AlphaTensor formulates matrix multiplication as a game where the agent needs to find the actions that achieve the correct result in the minimal amount of steps. Such RL algorithms are much more efficient than brute-force search even when purpose trained for a single problem(e.g., for each matrix size). The AlphaZero algorithm incorporates planning ahead with Monte - Carlo Tree Search to better predict the value of a state. The algorithm is very well suited for problems with a huge state space, such as chess or in our case all the different graph wiring configurations.

Another major improvement would be to use a better graph embedding, Perhaps using unsupervised methods. A better permutation - invariant embedding might result in an agent that could generalize to new unseen graphs and rewire them immediately with no extra training.

# References

Dsg (21/01/2020) *Understanding graph attention networks (GAT)*. Available at: https://dsgiitr.com/blogs/gat/ (Accessed: November 5, 2022).

*Papers with code - cora dataset*. Dataset | Papers With Code. (n.d.). Retrieved November 5, 2022, from https://paperswithcode.com/dataset/cora

Proximal Policy Optimization Algorithms John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov https://arxiv.org/abs/1707.06347